

سید محمد طاها طباطبایی – تمرین سری دوم

۹۸۱۲۷۶۲۸۳۸

چکیده:

در بخش اول، به بررسی و تحلیل هیستوگرام و متعادل سازی، تبدیل و اعمال توابعی که شدت روشنایی پیکسل ها را تحت تاثیر قرار می دهند، پرداختیم. در بخش دوم، به طور مشخص، به مقایسه تاثیر متعادل سازی کلی بر روی تصویر camera man پرداخته ایم. در پارت آخر، متعادل سازی محلی را روی ۴ نمونه تصویر مختلف و با سایز پنجره های متفاوت اعمال کردیم، تا بتوانیم تاثیر ابعاد پنجره مختلف و همچنین تاثیر متعادل سازی در تصاویر با نرخ جزئیات متفاوت را بررسی کنیم.

۲.۱.۱

توضیح فنی:

برای محاسبه هیستوگرام یک عکس مطابق عکس زیر، ابتدا با کمک لایبری OpenCV، عکس مورد نظر را لود می‌کنیم.

```
1 # load image
2 image = cv2.imread('Camera Man.bmp')
3 # calculate histogram
4 pdf = calc_histogram(image)
5 width , length , bands = image.shape
6
```

✓ 1.2s

سپس مطابق خط ۴، با استفاده از تابع ای که برای محاسبه هیستوگرام نوشته‌ام، هیستوگرام عکس را به‌دست می‌آوریم. تابع calc_histogram به شکل زیر تعریف می‌شود:

```
6 def calc_histogram(image):
7
8     length = image.shape[0]
9     width = image.shape[1]
10
11     pdf = np.zeros(256)
12
13     for i in range(length):
14         for j in range(width):
15             k = image[i][j]
16             pdf[k] +=1
17
18
19     return pdf
```

در این تابع، عکس مورد نظر به عنوان ورودی تابع استفاده می‌شود. در دو حلقه تو در تو، فراوانی رخ داد هر پیکسل که مقداری بین ۰ تا ۲۵۵ دارد را، در یک آرایه به نام pdf ذخیره می‌کنیم. طول این آرایه ۲۵۶ است، و اندیس k این آرایه، نشان دهنده فراوانی پیکسل‌هایی با مقدار k است.

```

1 # plt.style.use('_mpl-gallery')
2 plt.style.use(plt.style.available[5])
3
4 # plot histogram
5 fig, ax = plt.subplots(figsize=(17,8))
6 x = np.arange(0,256,1)
7 y = pdf
8 ax.stem(x,y)
9
10 ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
11 |      ylim=(0, 6501), yticks=np.arange(0, 6501,300))
12
13 plt.show()

```

✓ 0.2s

در گام آخر، مطابق قطعه کد بالا، با یک نمودار از نوع stem و با استفاده از کتابخانه matplotlib، نمودار هیستوگرام مربوط به عکس camera man را رسم می‌کنیم. در محور y این نمودار، مقادیر فراوانی یا همان pdf را جایگزاری می‌کنیم، و در محور x اعداد ۰ تا ۲۵۵.

نتیجه:

با اجرای این سلول‌ها، نمودار هیستوگرام عکس camera man را رسم می‌کنیم. در سه بخش این هیستوگرام تراکم پیکسل‌های را مشاهده می‌کنیم. بخش اول، از پیکسل‌هایی با مقدار ۲۰ تا اندکی پایین‌تر از ۱۰، که می‌توان دلیل آن را، وجود کت سیاه رنگ در تصویر باشد. همچنین یک رنج از پیکسل‌ها با مقادیر بین ۱۰۰ تا ۱۴۰ می‌توان به ناحیه‌ای که زمین را شامل می‌شود نسبت داد. رنج ۱۵۰ تا ۱۹۰ نیز می‌تواند نمایشگر بخش آسمان و فراوانی پیکسل‌های مربوط به این بخش باشد.

۲.۱.۱.۱

توضیح فنی:

در قطعه کدی مشابه تصویر زیر، مقدار هر پیکسل را بر ۳ تقسیم می‌کنیم. با اینکار انتظار داریم، تصویر تاریک‌تر شود. برای اینکار یک کپی از تصویر اصلی تهیه می‌کنیم و نام آن را D می‌گذاریم.

```
1 image = cv2.imread('Camera Man.bmp')
2
3 # decreasing brightness by dividing values by 3
4 D = np.copy(image)
5 for i in range(image.shape[0]):
6     for j in range(image.shape[1]):
7         D[i][j] = image[i][j]/3
8
```

نتیجه:

در بخش بعد بررسی می‌شود.

۲.۱.۱.۲

توضیح فنی:

```
1 # plot original image histogram
2 fig, ax = plt.subplots(figsize=(17,8))
3 x = np.arange(0,256,1)
4 y = pdf
5 ax.stem(x,y)
6
7 ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
8       ylim=(0, 6501), yticks=np.arange(0, 6501,300))
9
10 plt.show()
11
12 # plot D image histogram
13 D_pdf = calc_histogram(D)
14 fig, ax = plt.subplots(figsize=(17,12))
15 x = np.arange(0,256,1)
16 y = D_pdf
17 ax.stem(x,y)
18
19 ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
20       ylim=(0, 12001), yticks=np.arange(0, 12001,300))
21
22 plt.show()
```

✓ 1.9s

مطابق تصویر بالا، نمودار هیستوگرام عکس اصلی و عکس D را رسم می‌کنیم. هیستوگرام عکس D در خط ۱۳ محاسبه شده است. اجرای قطعه کد بالا، دو نمودار را به ترتیب ذکر شده رسم می‌کند.

نتیجه:

با اجرای سلول این بخش، در نمودار اول هیستوگرام عکس اصلی را رسم می کنیم که در بخش قبل نیز مشاهده شد. در نمودار دوم که مربوط به هیستوگرام عکس D است، تفاوت فاحشی بین فراوانی پیکسل ها با عکس اصلی داریم. فراوانی پیکسل های بخش های روشن تصویر، به طور معنی داری، کاهش یافته، و فراوانی پیکسل های بخش های تاریک تر تصویر، بسیار بیشتر شده است. به طور مثال، رنج پیک ۱۵۰ تا ۱۹۰ در عکس قبلی، به حدود ۵۰ تا ۶۵ مپ شده است و فراوانی این پیکسل ها بسیار زیادتر از عکس اول شده است.

۲.۱.۱.۳

توضیح فنی:

```
1 # first, we should normalize the pdf
2 normal_D_pdf = np.zeros(len(D_pdf))
3 normal_D_pdf = normalizeHistogram(D_pdf,D.shape[0],D.shape[1])
4 # calculating cdf
5 D_cdf = np.zeros(len(D_pdf))
6 D_cdf = calc_cdf(normal_D_pdf)
7 # re-map original image pixels to new ones
8 H = reMap(image,D_cdf)
9
10 plt.imshow(H)
11
12
✓ 0.5s
```

برای اجرای روال متعادل سازی هیستوگرام، در ابتدا نیاز داریم تا طبق الگوریتم ارائه شده در درس، هیستوگرام فعلی را نرمالایز کنیم. برای اینکار یک تابع به نام normalizeHistogram نوشته ایم.

```
21 def normalizeHistogram(pdf,width,length):
22     normal_pdf = np.zeros(len(pdf))
23
24     for i in range(len(normal_pdf)):
25         normal_pdf[i] = pdf[i]/(width*length)
26
27     return normal_pdf
```

در این تابع، مقادیر pdf را به رنج بین ۰ تا ۱ مپ می کنیم. چون ماکسیمم مقدار فراوانی برابر حاصل ضرب طول در عرض تصویر است، برای تقسیم کردن در مخرج از این عدد استفاده می کنیم.

در گام دوم، مطابق الگوریتم به محاسبه cdf (تابع چگالی) می پردازیم. برای اینکار تابع calc_cdf را تعریف می کنیم.

```

29 def calc_cdf(normal_pdf):
30
31     cdf = np.zeros(len(normal_pdf))
32
33     for i in range(len(normal_pdf)):
34         for j in range(i):
35             cdf[i] += normal_pdf[j]
36
37     cdf[i] *=255
38     cdf[i] = round(cdf[i])
39
40     return cdf

```

در این تابع، به ازای مقدار i ام از آرایه نرمال شده، تمام اندیس های ۰ تا i از آرایه نرمال شده را با هم جمع می کنیم. طبق فرمول محاسبه cdf، در حلقه بیرونی، که در واقع انتهای سیگمای جمع مقادیر نرمالایز شده تا آن اندیس است، مقدار آن اندیس را در ۲۵۵ ضرب می کنیم. در انتها چون این مقدار ممکن است یک عدد اعشاری باشد، آن را به مقدار صحیح نزدیک تر به آن گرد می کنیم.

در گام نهایی نیاز است تا با توجه به تابع تبدیل به دست آمده، مقادیر پیکسل های عکس را آپدیت کنیم. برای اینکار، از یک تابع به نام reMap نوشته ایم.

```

42 def reMap(image,target_cdf):
43     newImage = image.copy()
44     for i in range(newImage.shape[0]):
45         for j in range(newImage.shape[1]):
46             newImage[i][j] = target_cdf[newImage[i][j]]
47
48     return newImage
49

```

این تابع با دریافت عکس هدف و cdf مورد نظر، پیکسل های عکس را با مقادیر جدید بر اساس cdf داده شده، آپدیت می کند.

نتیجه:

اگر نتیجه این تبدیل را مشاهده کنیم، می بینیم که عکس دارای قسمت های بسیار سفید است، که به دلیل وجود شیب زیاد در تابع تبدیل به دلیل وجود قله هایی از فراوانی پیکسل، در هیستوگرام عکس ورودی است. یک روش حل این مشکل، برش زدن هیستوگرام است، که چون در صورت سوال رفع این مشکل خواسته نشده، رفع مشکل پیاده سازی نشده است. روش دیگر حل این مشکل، استفاده از متعادل سازی محلی است، که در بخش بعدی پیاده سازی شده است.

۲.۱.۱.۴

توضیح فنی:

```

1 L = local_histo_equalization(D,32)
2 plt.imshow(L)

```

✓ 4.4s

برای پیاده سازی متعادل سازی محلی هیستوگرام، یک تابع پیاده سازی کرده ایم.

```

63  def local_histo_equalization(image,windowSize):
64      width , length , band =image.shape
65
66      newImage = np.zeros_like(image)
67      n = 0
68      for i in range(0,length,windowSize):
69          for j in range(0,width,windowSize):
70              start = (j,i)
71              end0 = j+windowSize
72              end1 = i+windowSize
73
74              if end0>width:
75                  end0 = width
76
77              if end1>length:
78                  end1 = length
79
80              end = (end0,end1)
81
82              buffer = image[start[0]:end[0], start[1]:end[1]]
83

```

ورودی این تابع، یک عکس و ابعاد پنجره مورد نظر برای بخش‌بندی تصویر بر اساس این پنجره و محاسبات به صورت محلی در این پنجره است. مطابق شکل، ابتدا براساس اندازه پنجره، روی عکس حرکت می‌کنیم و یک برش با اندازه $windowSize*windowSize$ از عکس را انتخاب می‌کنیم. متغیر بافر، یک متغیر میانی است که ابتدا محاسبات را برای هر برش از عکس، روی این تصویر انجام می‌دهیم، سپس نتایج را به متغیر مورد نظر برای عکس جدید منتقل می‌کنیم.

```

83
84      buffer_histo = calc_histogram(buffer)
85      normal_buffer = normalizeHistogram(buffer_histo,windowSize,windowSize)
86      buffer_cdf = calc_cdf(normal_buffer)
87      remaped_buffer = reMap(buffer,buffer_cdf)
88
89      newImage[start[0]:end[0], start[1]:end[1]] = remaped_buffer
90
91      n +=1
92
93      eqaulizedImage = newImage
94      return eqaulizedImage

```

در ادامه، مطابق حالت عادی متعادل سازی، برای هر بخش، روال متعادل سازی را روی برش انتخاب شده، اعمال می‌کنیم. در انتهای حلقه، بافر ریمپ شده را، در یک آرایه با نام `newImage` که درواقع عکس تولیدی جدید است، جایگزاری می‌کنیم.

نتیجه:

با مشاهده عکس تولیدی در این مرحله، می‌بینیم که عکس تولیدی در بخشی نقاط حالت پنجره پنجره دارد. دلیل این اتفاق، این است که در پنجره‌هایی که از نقاط یکدست تصویر نمونه برداری شده است، فقط فراوانی برخی مقادیر خاص از رنج ۰ تا ۲۵۵ زیاد است، و فراوانی بقیه مقادیر نزدیک به صفر است، و وجود این پیک‌ها متعادل سازی را کمی با مشکل مواجه می‌کند. اما در بخش‌هایی که پنجره‌ها، از قسمت‌هایی نمونه برداری کرده‌اند که شامل جزئیات عکس است، این حالت پنجره پنجره شدن کمتر دیده می‌شود و تا حدود زیادی، عملکرد محلی سازی بهتر از متعادل سازی کلی است، چون باعث می‌شود تا کنتراست بهتر تغییر کند و سایه‌ها و لبه‌ها نیز مشخص‌تر هستند.

۲.۱.۱.۵

توضیح فنی:

هیستوگرام های H و L را محاسبه و رسم می کنیم. میله های آبی رنگ روی نمودار، نشان دهنده مقادیر هیستوگرام تصویر H و میله های قرمز مربوط به تصویر L است.

```
1 # plot H image histogram (blue)
2 fig, ax = plt.subplots(figsize=(15,8))
3 x = np.arange(0,256,1)
4 y = calc_hitogram[H]
5 ax.stem(x,y)
6
7 # plot L image histogram (red)
8 y = calc_hitogram[L]
9 ax.stem(x,y)
10 markerline ,stemlines ,baseline = ax.stem(x,y,linefmt='red',markerfmt='x')
11 markerline.set_color('red')
12
13 ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
14        ylim=(0, 6001), yticks=np.arange(0, 6001,200))
15
16 plt.show()
```

✓ 2.8s

نتیجه:

مطابق نمودار های رسم شده، میبینیم که هیستوگرام تصویر L، نسبت به تصویر H از پراکندگی بیشتری برخوردار است. مطابق انتظار، هیستوگرام تصویر H، به دلیل اینکه شامل دو ناحیه سفید و سیاه است، به همین شکل باشد، یعنی در رنج بالاتر از ۲۵۰ و رنج ۵۰ تا ۶۰، فراوانی مقادیر بسیار زیاد است و در بقیه مقادیر، فراوانی ها کم یا صفر است. برعکس در تصویر L، نموداری داریم که تقریباً فراوانی پراکنده ای دارد، و نمی توان یک قله مشخص و برجسته را در آن مشخص کرد، علی رغم اینکه در برخی مقادیر، یک پرش و افزایش فراوانی، نسبت به مقادیر همسایه خود دارد، که قابل چشم پوشی است.

۲.۱.۱.۶

توضیح فنی:

در سلول اول این بخش، تبدیل لگاریتمی مطابق فرمول ارائه شده در اسلاید درس پیاده سازی شده است. در ادامه نیز تصویر حاصل و هیستوگرام آن رسم شده است.

```
1 # log transform
2 c = 255 / np.log(1 + np.max(image))
3 D_log = log_transform(D,c)
4 D_log = np.array(D_log, dtype = np.uint8)
```

پیاده سازی تابع log_transform به شکل زیر است:

```
49
50 def log_transform(image, c):
51     s = c*np.log(1+image)
52     return s
```


در سلول دوم، تبدیل معکوس لگاریتمی پیاده‌سازی شده است.

```
1 # inverse log transform
2 base = 2
3 c = 255.0 / (pow(base, np.max(image)) - 1)
4 D_iLog = inverse_log_transform(D,c,base)
5 D_iLog = np.array(D_iLog, dtype = np.uint8)
```

پیاده‌سازی inverse_log_transform:

```
54 def inverse_log_transform(image , c, base):
55     s = c*(np.power(base,image)-1)
56     return s
```

در سلول سوم، تبدیل نمایی پیاده‌سازی شده است.

```
1 # power-law transform
2 gamma = 1.5
3 # c = 255 / np.power(image,gamma)
4 c = 2
5 D_power = power_law_transform(D,c,gamma)
6 D_power = np.array(D_power, dtype = np.uint8)
```

پیاده‌سازی power_law_transform:

```
58 def power_law_transform(image, c, gamma):
59     s = c*np.power(image,gamma)
60     return s
```

نتیجه:

مقادیر پارامترها برای توابع، با کمک و مطالعه [این لینک](#) و [این لینک](#) انتخاب شده است.

۲.۱.۲

توضیح فنی:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import cv2
4 from functions import *
5
6 cameraman = cv2.imread("Camera Man.bmp")
7 width , length ,band = cameraman.shape
8 cm_histogram = calc_histogram(cameraman)
9 cm_histogram_normal = normalizeHistogram(cm_histogram,width,length)
10 cm_cdf = calc_cdf(cm_histogram_normal)
11 newCameraMan = reMap(cameraman,cm_cdf)
12 ncm_histogram = calc_histogram(newCameraMan)
13
```

مطابق تصویر و روالی که در قسمت ۲.۱.۱.۳ نیز توضیح داده شده، عکس camera man را لود کردیم و مراحل متعادل سازی را روی آن اجرا کرده ایم. در ابتدا هیستوگرام عکس با تابع calc_histogram محاسبه شده، سپس با تابع normalizeHistogram هیستوگرام به دست آمده را نرمالایز کرده ایم. در گام بعد، cdf را با تابع calc_cdf محاسبه کرده ایم. در نهایت نیز بر اساس cdf محاسبه شده، عکس جدید camera man را ساخته ایم. در گام نهایی نیز هیستوگرام عکس جدید را حساب کرده ایم.

```
14
15 figure = plt.figure(figsize=(10,10))
16
17 figure.add_subplot(1,2,1)
18 plt.imshow(newCameraMan)
19 plt.title("equalized",color='black')
20
21 figure.add_subplot(1,2,2)
22 plt.imshow(cameraman)
23 plt.title("original",color='black')
24
```

قطعه کد نمایش عکس اصلی و عکس متعادل شده.

```
26 # plot original image histogram
27 fig, ax = plt.subplots(figsize=(15,5))
28 x = np.arange(0,256,1)
29 y = cm_histogram
30 ax.stem(x,y)
31
32 y = ncm_histogram
33
34 markerline ,stemlines ,baseline = ax.stem(x,y,linefmt='red',markerfmt='x')
35 markerline.set_color('red')
36
37 ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
38       ylim=(0, 6601), yticks=np.arange(0, 6601,200))
39
40 plt.show()
41
```

قطعه کد رسم نمودار هیستوگرام عکس اصلی و عکس جدید.

نتیجه:

مطابق انتظار، کنتراست تصویر بهبود یافته است، پراکندگی فراوانی در نمودار متعادل شده بیشتر است و تفاوت بین لبه های سیاه و سفید واضح تر شده است.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import cv2
4 from functions import *
5
6 he1 = cv2.imread('HE1.jpg')
7 he2 = cv2.imread('HE2.jpg')
8 he3 = cv2.imread("HE3.jpg")
9 he4 = cv2.imread('HE4.jpg')
10

```

قطعه کد لود کردن تصاویر.

```

11 local_eq1_he1 = local_histo_equalization(he1,16)
12 local_eq2_he1 = local_histo_equalization(he1,32)
13 local_eq3_he1 = local_histo_equalization(he1,128)
14
15 local_eq1_he2 = local_histo_equalization(he2,16)
16 local_eq2_he2 = local_histo_equalization(he2,32)
17 local_eq3_he2 = local_histo_equalization(he2,128)
18
19
20 local_eq1_he3 = local_histo_equalization(he3,16)
21 local_eq2_he3 = local_histo_equalization(he3,32)
22 local_eq3_he3 = local_histo_equalization(he3,128)
23
24 local_eq1_he4 = local_histo_equalization(he4,16)
25 local_eq2_he4 = local_histo_equalization(he4,32)
26 local_eq3_he4 = local_histo_equalization(he4,128)
27

```

قطعه کد پردازش تصاویر به صورت محلی، به ترتیب با پنجره هایی با سایز ۱۶، ۳۲ و ۱۲۸.

در ادامه قطعه کد این سلول نیز تصاویر متعادل شده، نمایش داده شده اند.

نتیجه:

در بحث تاثیر ابعاد پنجره، در پنجره های بسیار کوچک، عکس یک حالت پنجره-پنجره شدن پیدا می کند، و چون هر پنجره به مقادیر ۰ تا ۲۵۵ اسکیل می شود، در هر پنجره کوچک، نقاط بسیار سفید و بسیار تیره پدید می آید، که در حالی که برای تصاویر شلوغ، شاید باعث کنتراست بهتر شود، اما قطعا برای نواحی یکدست (فرکانس تغییرات کم) باعث پدیدار شدن حالت نویزی می شود. در سمت مقابل، پنجره بزرگتر، شاید اثر نویزی کمتری داشته باشد، اما در نواحی که تغییرات بسیار شدید است، نسبت به پنجره با ابعاد وچکتر، کنتراست کمتری ایجاد می کند. بنابر توضیح بالا، باید با توجه به هر تصویر، تصمیم گرفت که از چه سایز پنجره ای برای متعادل سازی محلی استفاده کرد. با توجه به تصاویر داده شده، سایز پنجره ۱۲۸ یا ۶۴ حالت مناسبی است، که هم کنتراست خوبی ایجاد می کند، و هم عکس حالت نویزی پیدا نمی کند.

۲.۱.۳

توضیح فنی:

تابع `histeq`، با کمک متعادل‌سازی هیستوگرام، شدت روشنایی پیکسل‌های تصویر را تغییر می‌دهد، در حالی که تابع `imadjust`، شدت روشنایی پیکسل‌ها را، فارغ از متعادل‌سازی هیستوگرام انجام می‌دهد.

نتیجه:

ندارد

2.1. Histogram Equalization

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
import cv2
from functions import *
```

2.1.1. Write a program that can compute the histogram of a grayscale image (assuming 256 levels of gray). In a separate main program, apply the program to Camera Man image, and illustrate the histogram as a stem plot besides the test image (using "subplot" function).

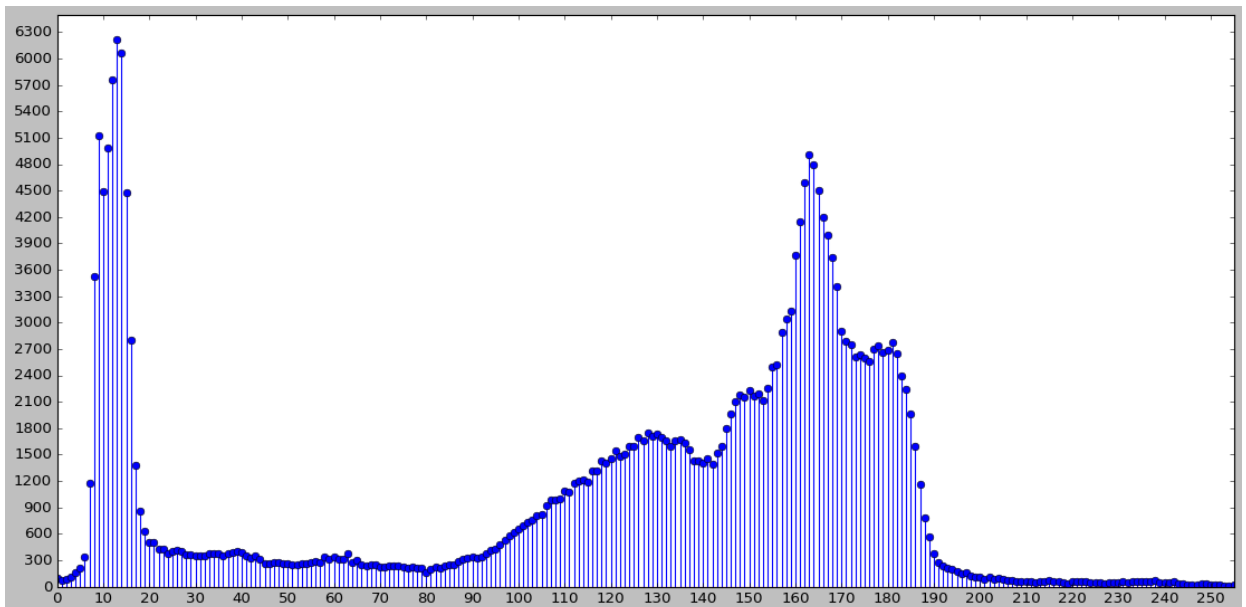
```
In [ ]: # Load image
image = cv2.imread('Camera Man.bmp')
# calculate histogram
pdf = calc_histogram(image)
width , length , bands = image.shape
```

```
In [ ]: # plt.style.use('_mpl-gallery')
plt.style.use(plt.style.available[5])

# plot histogram
fig, ax = plt.subplots(figsize=(17,8))
x = np.arange(0,256,1)
y = pdf
ax.stem(x,y)

ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
       ylim=(0, 6501), yticks=np.arange(0, 6501,300))

plt.show()
```



2.1.1.1. Decrease the brightness of Camera Man by dividing the intensity values by 3 and named output as D.

```
In [ ]: image = cv2.imread('Camera Man.bmp')
```

```

# decreasing brightness by dividing values by 3
D = np.copy(image)
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        D[i][j] = image[i][j]/3

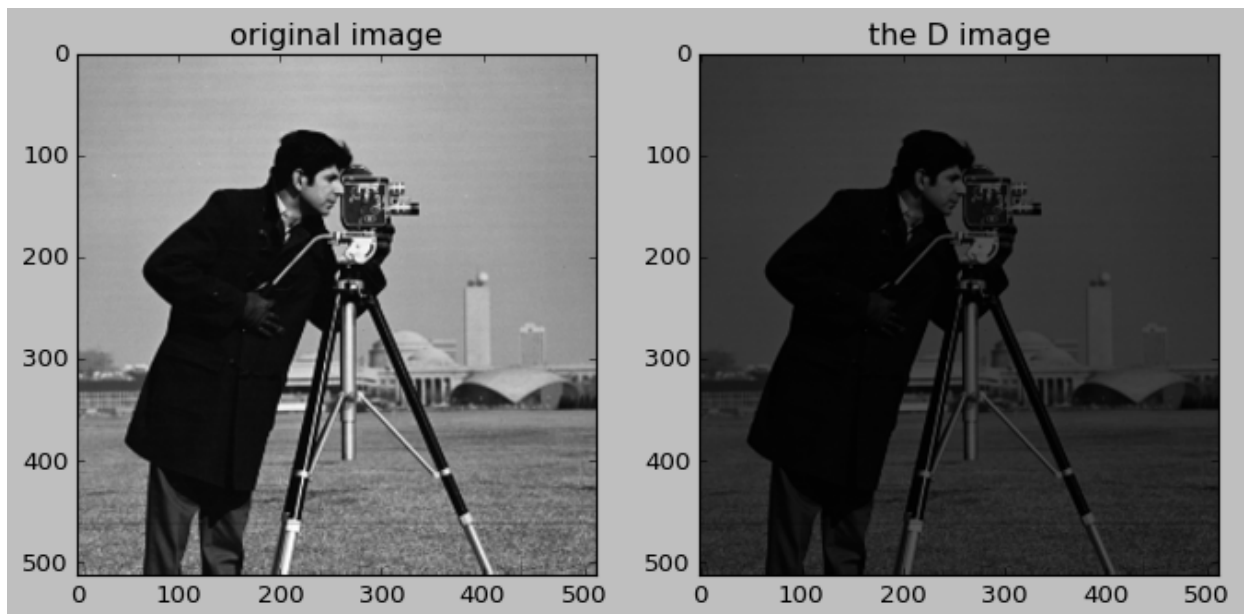
figure = plt.figure(figsize=(10,10))
plt.style.use(plt.style.available[5])

figure.add_subplot(1,2,1)
plt.imshow(image)
plt.title('original image',color='black')
# plt.show()

figure.add_subplot(1,2,2)
plt.imshow(D)
plt.title('the D image',color='black')
# plt.show()

```

Out[]: Text(0.5, 1.0, 'the D image')



2.1.1.2. Plot the histograms of Input and D. What can you observe from these two histograms?

```

In [ ]: # plot original image histogram
fig, ax = plt.subplots(figsize=(17,8))
x = np.arange(0,256,1)
y = pdf
ax.stem(x,y)

ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
       ylim=(0, 6501), yticks=np.arange(0, 6501,300))

plt.show()

# plot D image histogram
D_pdf = calc_histogram(D)
fig, ax = plt.subplots(figsize=(17,12))
x = np.arange(0,256,1)

```

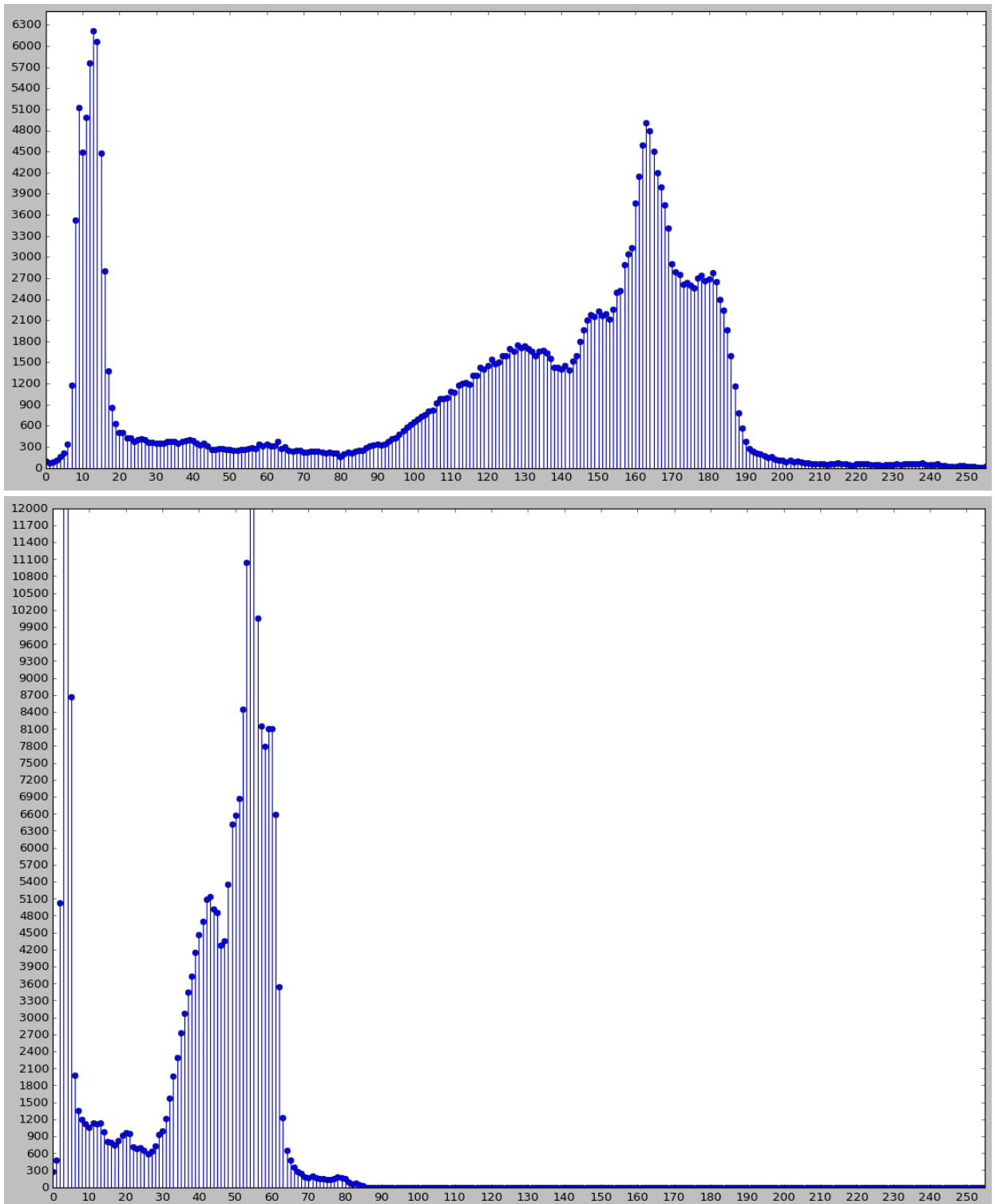
```

y = D_pdf
ax.stem(x,y)

ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
      ylim=(0, 12001), yticks=np.arange(0, 12001,300))

plt.show()

```



2.1.1.3. Perform histogram equalization on D and output the result as H.

```

In [ ]: # first, we should normalize the pdf
normal_D_pdf = np.zeros(len(D_pdf))

```



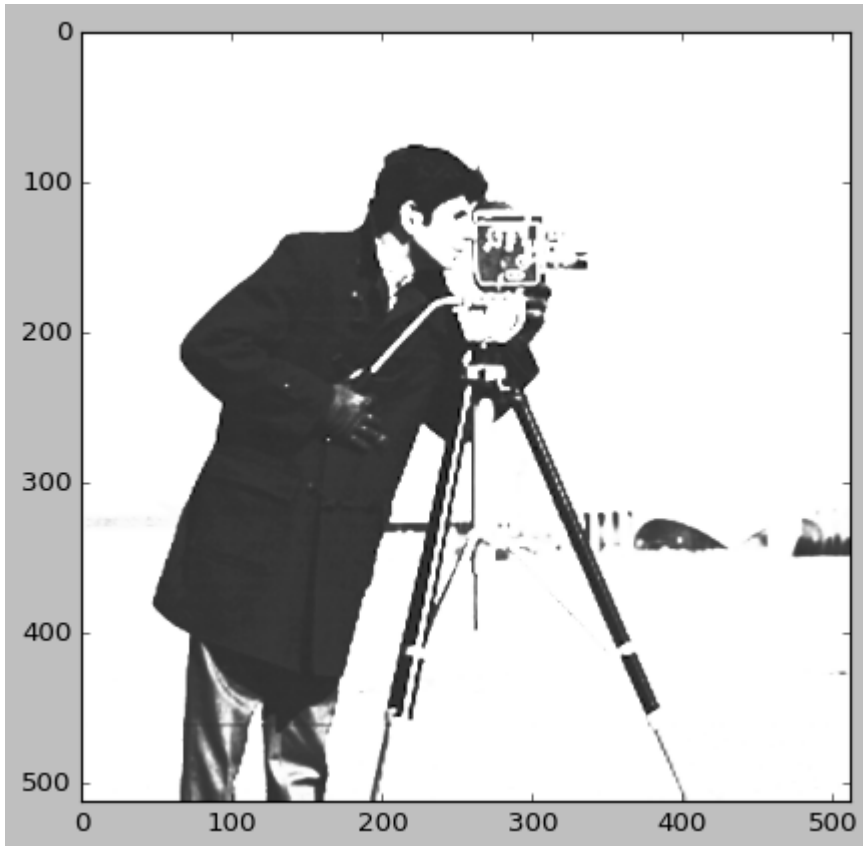
```

normal_D_pdf = normalizeHistogram(D_pdf,D.shape[0],D.shape[1])
# calculating cdf
D_cdf = np.zeros(len(D_pdf))
D_cdf = calc_cdf(normal_D_pdf)
# re-map original image pixels to new ones
H = reMap(image,D_cdf)

plt.imshow(H)

```

Out[]: <matplotlib.image.AxesImage at 0x25c9c689ed0>



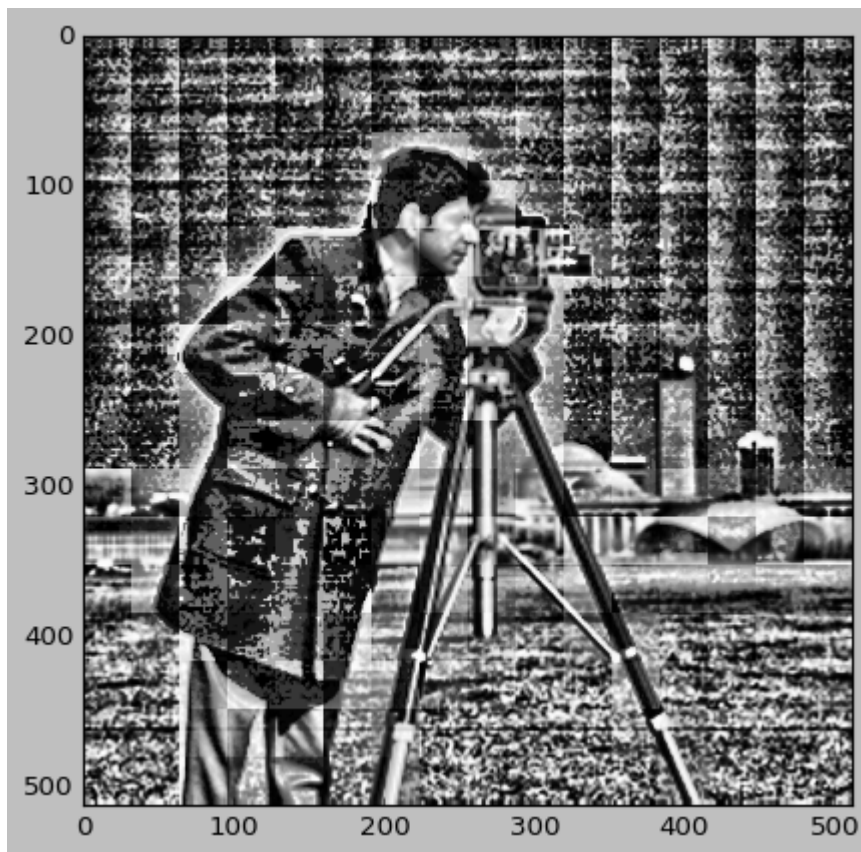
2.1.1.4. Perform local histogram equalization on image D and output the result as L.

```

In [ ]: L = local_histo_equalization(D,32)
plt.imshow(L)

```

Out[]: <matplotlib.image.AxesImage at 0x25c9c6f0e50>



2.1.1.5. Plot the histograms of H and L. What's the main difference between local and global histogram equalization?

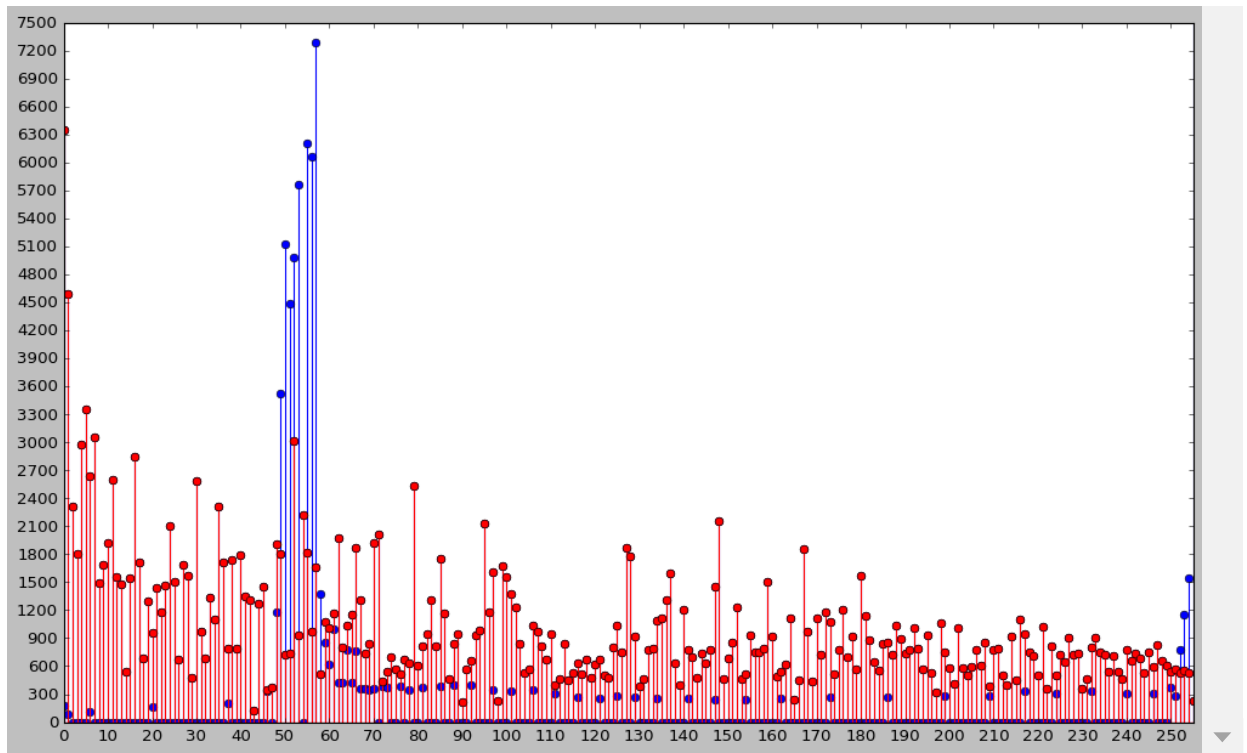
```
In [ ]: # plot H image histogram (blue)
fig, ax = plt.subplots(figsize=(15,9))
x = np.arange(0,256,1)
y = calc_histogram(H)
print(H.shape)
ax.stem(x,y)

# plot L image histogram (red)
y = calc_histogram(L)
print(L.shape)
ax.stem(x,y)
markerline ,stemlines ,baseline = ax.stem(x,y,linewidth='red',markerfmt='o')
markerline.set_color('red')

ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
       ylim=(0, 7501), yticks=np.arange(0, 7501,300))

plt.show()
```

```
(512, 512, 3)
(512, 512, 3)
```



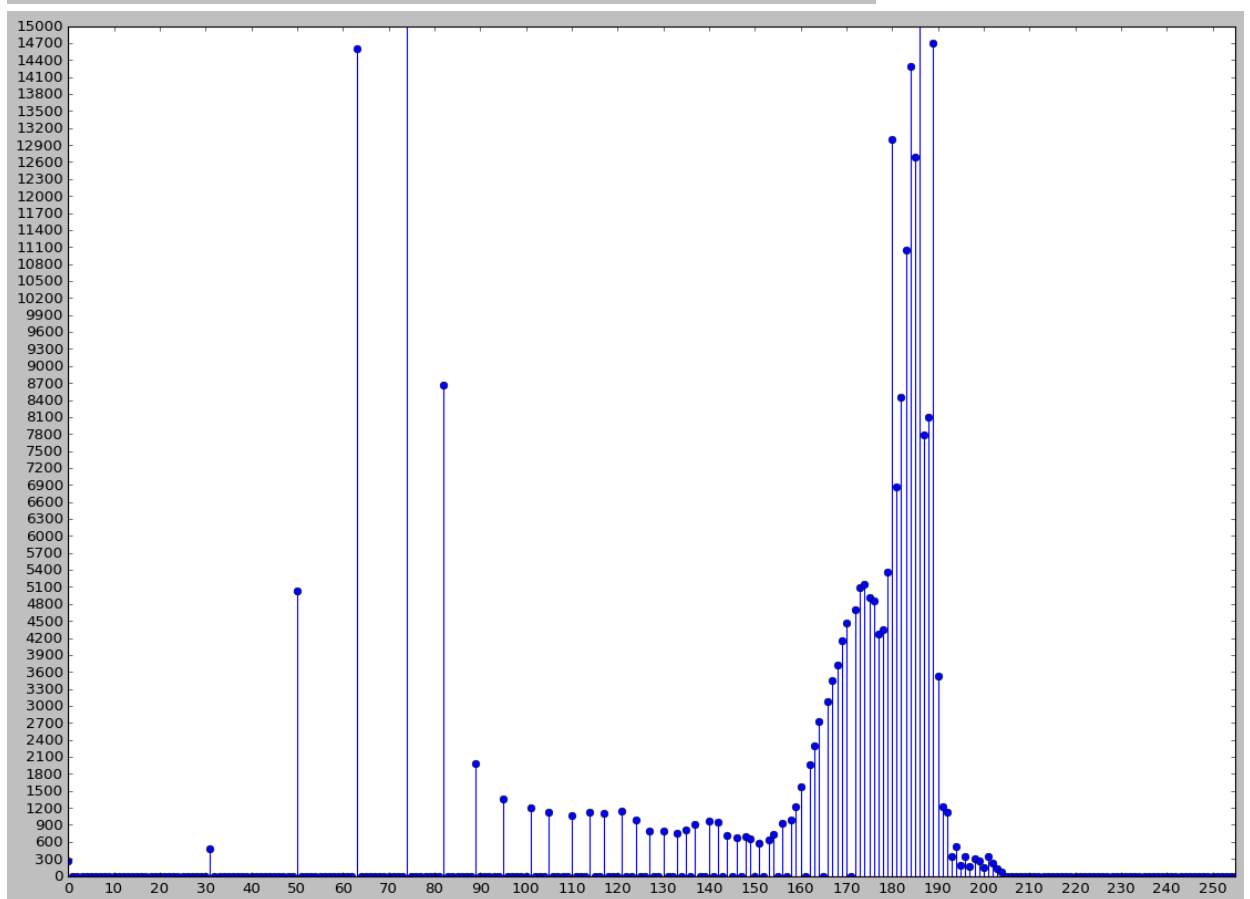
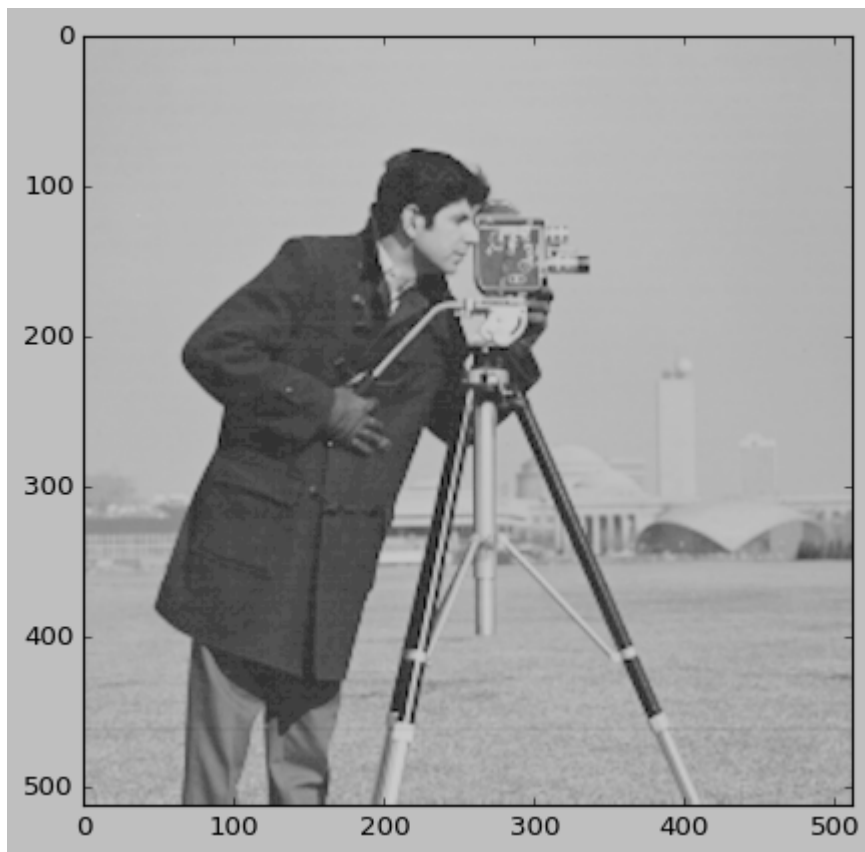
2.1.1.6. Perform the log transform, inverse log transform and power-law transform to enhance image D. Please adjust the parameters to obtain the results as best as you can. Show the parameters, resultant images and corresponding histograms. Provide some discussions on the results as well.

```
In [ ]: # Log transform
c = 255 / np.log(1 + np.max(image))
D_log = log_transform(D,c)
D_log = np.array(D_log, dtype = np.uint8)
plt.imshow(D_log)
plt.show()

# plot D_Log image histogram
D_log_pdf = calc_histogram(D_log)
fig, ax = plt.subplots(figsize=(17,12))
x = np.arange(0,256,1)
y = D_log_pdf
ax.stem(x,y)

ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
       ylim=(0, 15001), yticks=np.arange(0, 15001,300))

plt.show()
```



```
In [ ]: # inverse log transform
base = 2
c = 255.0 / (pow(base, np.max(image)) - 1)
# c = 2
D_iLog = inverse_log_transform(D,c,base)
```

```

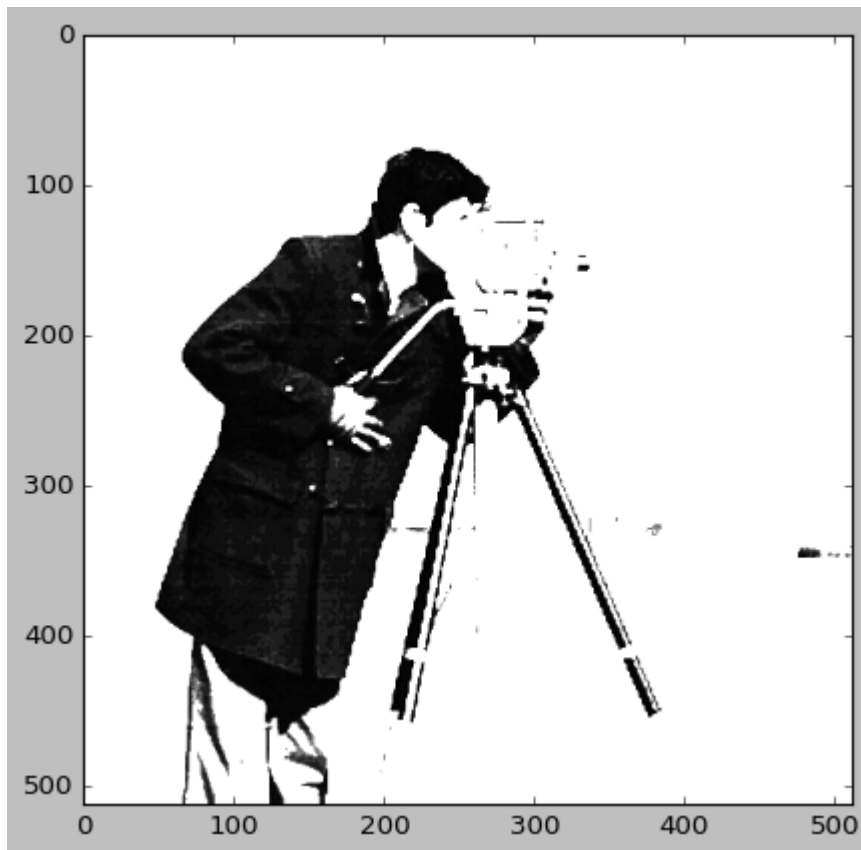
D_iLog = np.array(D_iLog, dtype = np.uint8)
plt.imshow(D_iLog)
plt.show()

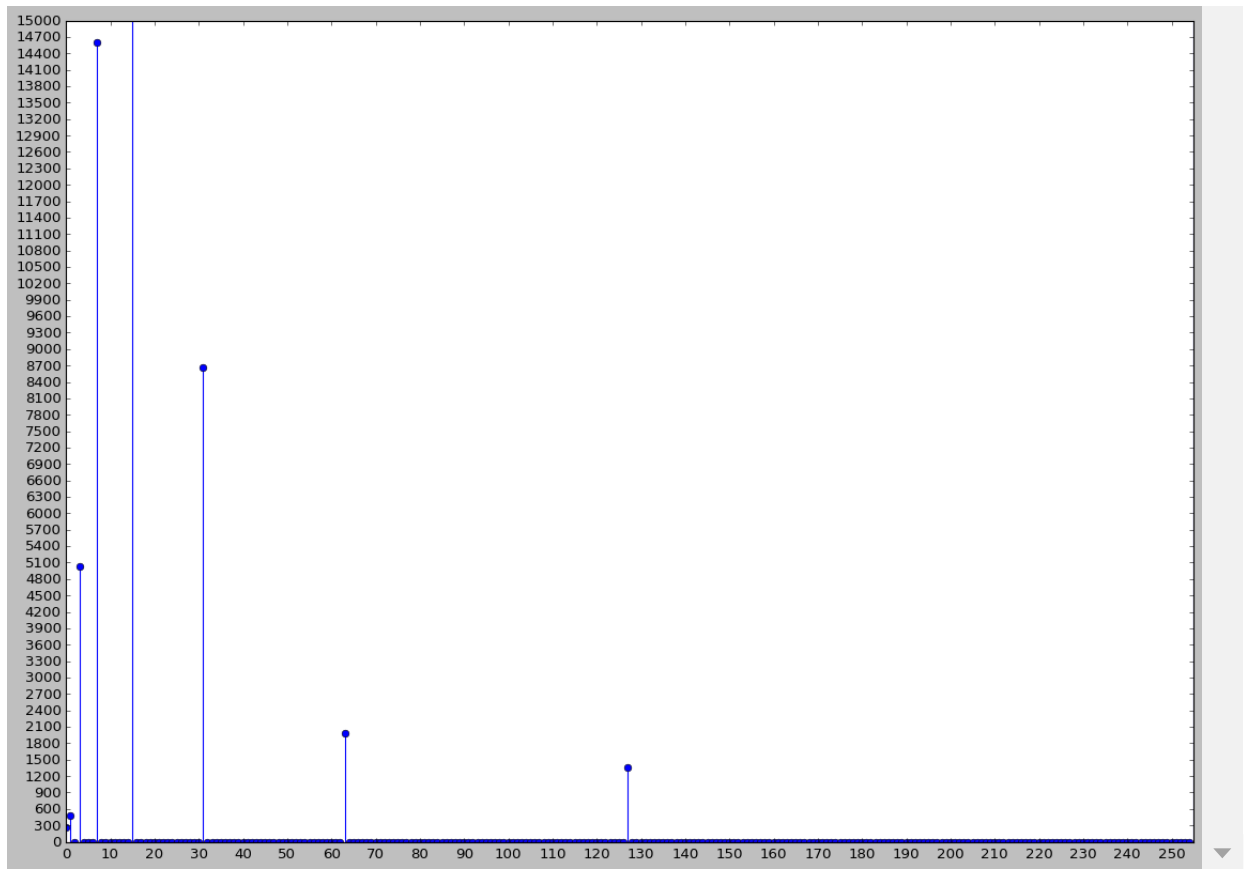
# plot D_iLog image histogram
D_iLog_pdf = calc_histogram(D_iLog)
fig, ax = plt.subplots(figsize=(17,12))
x = np.arange(0,256,1)
y = D_iLog_pdf
ax.stem(x,y)

ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
       ylim=(0, 15001), yticks=np.arange(0, 15001,300))

plt.show()

```



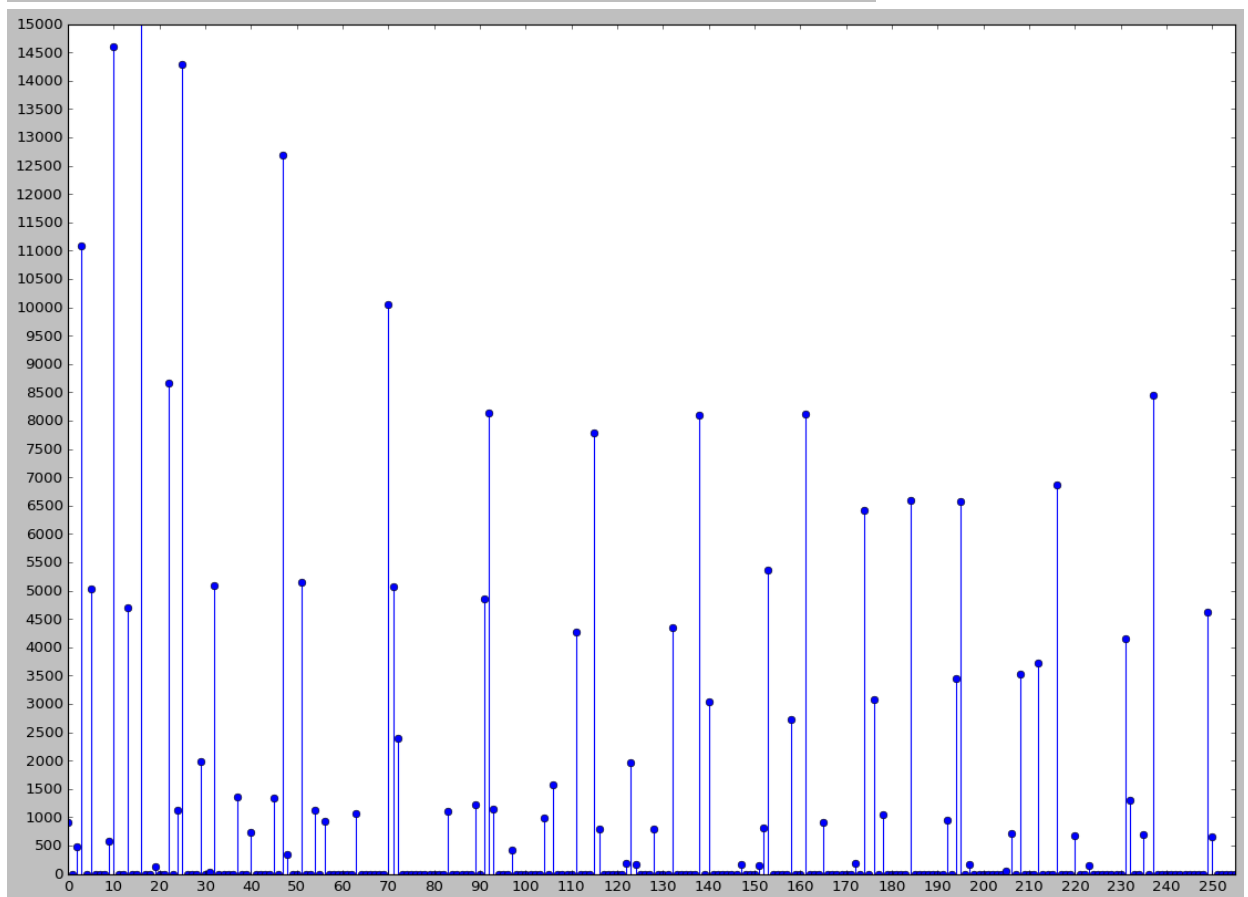
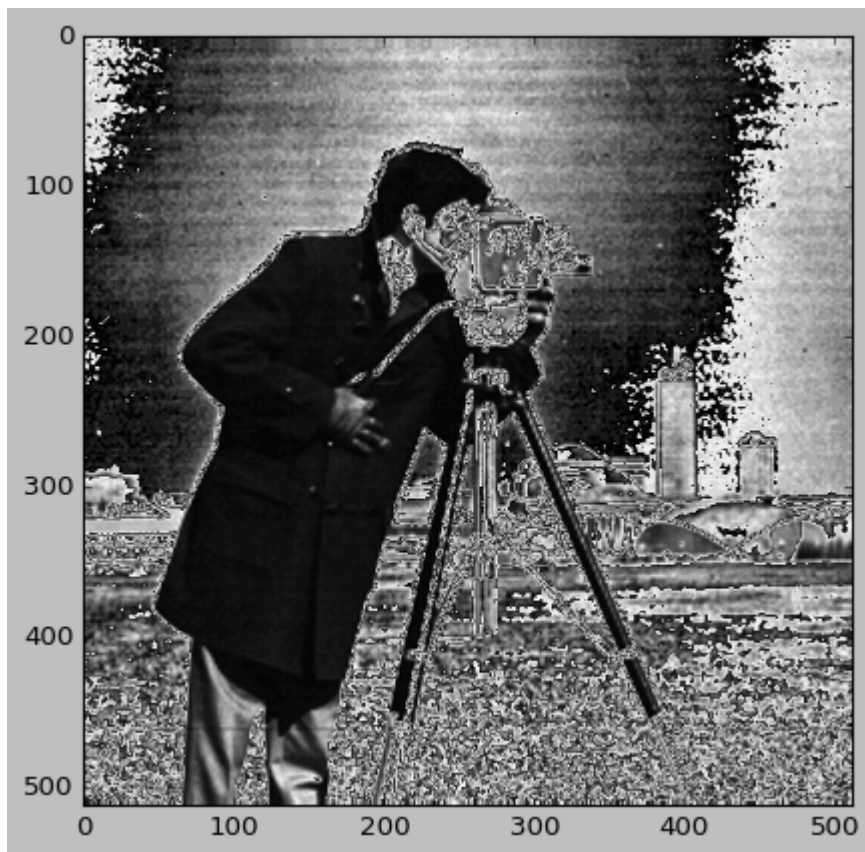


```
In [ ]: # power-law transform
gamma = 1.5
# c = 255 / np.power(image, gamma)
c = 2
D_power = power_law_transform(D, c, gamma)
D_power = np.array(D_power, dtype = np.uint8)
plt.imshow(D_power)

# plot D_power image histogram
D_power_pdf = calc_histogram(D_power)
fig, ax = plt.subplots(figsize=(17,12))
x = np.arange(0, 256, 1)
y = D_power_pdf
ax.stem(x, y)

ax.set(xlim=(0, 255), xticks=np.arange(0, 255, 10),
       ylim=(0, 15001), yticks=np.arange(0, 15001, 500))

plt.show()
```



2.1.2. Write a program that performs histogram equalization on Camera Man image. Display the original and equalized images, as well as their corresponding histograms, all in one figure as mentioned in 2.1.1.

```

In [ ]: import numpy as np
import matplotlib.pyplot as plt
import cv2
from functions import *

cameraman = cv2.imread("Camera Man.bmp")
width , length ,band = cameraman.shape
cm_histogram = calc_histogram(cameraman)
cm_histogram_normal = normalizeHistogram(cm_histogram,width,length)
cm_cdf = calc_cdf(cm_histogram_normal)
newCameraMan = reMap(cameraman,cm_cdf)
ncm_histogram = calc_histogram(newCameraMan)

figure = plt.figure(figsize=(10,10))

figure.add_subplot(1,2,1)
plt.imshow(newCameraMan)
plt.title("equalized",color='black')

figure.add_subplot(1,2,2)
plt.imshow(cameraman)
plt.title("original",color='black')

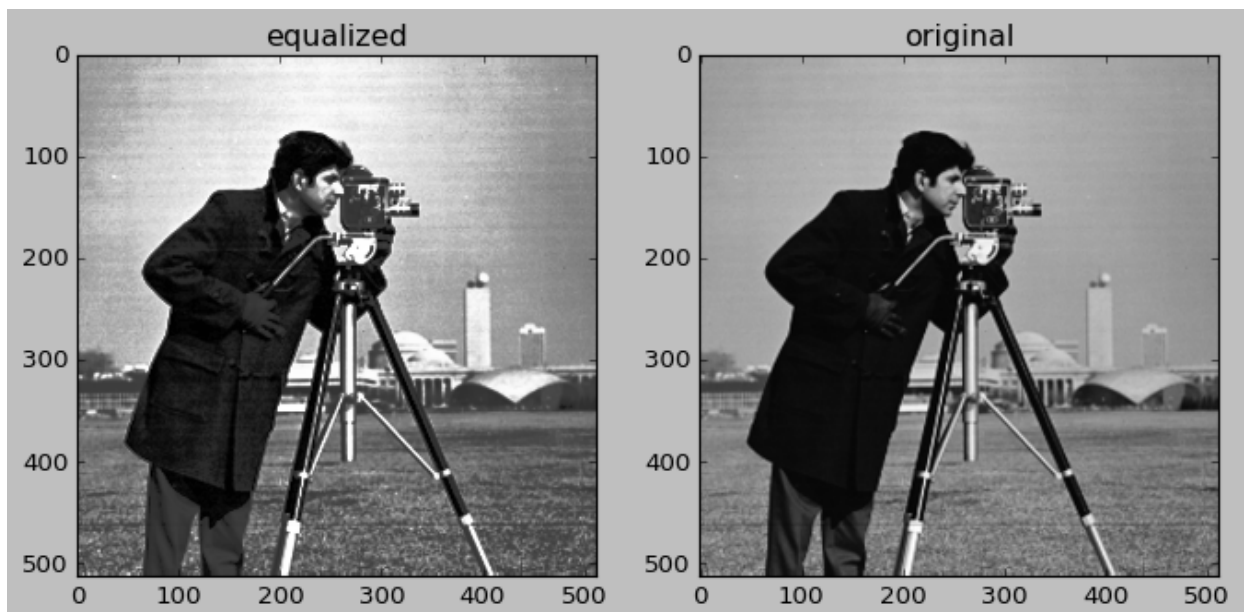
# plot original image histogram
fig, ax = plt.subplots(figsize=(15,5))
x = np.arange(0,256,1)
y = cm_histogram
ax.stem(x,y)

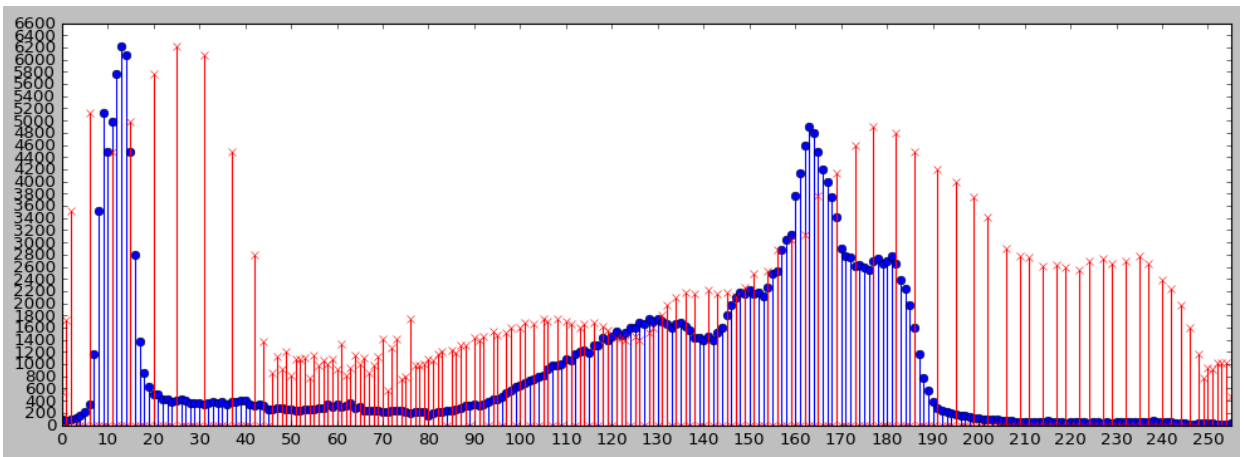
y = ncm_histogram

markerline ,stemlines ,baseline = ax.stem(x,y,linewidth='red',markerfmt='x')
markerline.set_color('red')

ax.set(xlim=(0, 255), xticks=np.arange(0, 255,10),
      ylim=(0, 6601), yticks=np.arange(0, 6601,200))
plt.style.use(plt.style.available[5])
plt.show()

```





2.2. Local Histogram Equalization

2.2.1. Implement a local histogram equalization with different windows size for the HE1,2,3, and 4 images. Explain and display the results. Discuss the effects of increasing window size and compare it with global histogram equalization in detail.

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
import cv2
from functions import *

he1 = cv2.imread('HE1.jpg')
he2 = cv2.imread('HE2.jpg')
he3 = cv2.imread("HE3.jpg")
he4 = cv2.imread('HE4.jpg')

local_eq1_he1 = local_histo_equalization(he1,16)
local_eq2_he1 = local_histo_equalization(he1,32)
local_eq3_he1 = local_histo_equalization(he1,128)

local_eq1_he2 = local_histo_equalization(he2,16)
local_eq2_he2 = local_histo_equalization(he2,32)
local_eq3_he2 = local_histo_equalization(he2,128)

local_eq1_he3 = local_histo_equalization(he3,16)
local_eq2_he3 = local_histo_equalization(he3,32)
local_eq3_he3 = local_histo_equalization(he3,128)

local_eq1_he4 = local_histo_equalization(he4,16)
local_eq2_he4 = local_histo_equalization(he4,32)
local_eq3_he4 = local_histo_equalization(he4,128)

figure = plt.figure(figsize=(8,8))
figure.add_subplot(2,2,1)
plt.imshow(local_eq1_he1)
plt.title("window 16",color='black')

figure.add_subplot(2,2,2)
plt.imshow(local_eq2_he1)
plt.title("window 32",color='black')

figure.add_subplot(2,2,3)
```

```

plt.imshow(local_eq3_he1)
plt.title("window 128",color='black')

figure.add_subplot(2,2,4)
plt.imshow(he1)
plt.title("he1",color='black')

figure = plt.figure(figsize=(8,8))
figure.add_subplot(2,2,1)
plt.imshow(local_eq1_he2)
plt.title("window 16",color='black')

figure.add_subplot(2,2,2)
plt.imshow(local_eq2_he2)
plt.title("window 32",color='black')

figure.add_subplot(2,2,3)
plt.imshow(local_eq3_he2)
plt.title("window 128",color='black')

figure.add_subplot(2,2,4)
plt.imshow(he2)
plt.title("he2",color='black')
plt.show()

figure = plt.figure(figsize=(8,8))
figure.add_subplot(2,2,1)
plt.imshow(local_eq1_he3)
plt.title("window 16",color='black')

figure.add_subplot(2,2,2)
plt.imshow(local_eq2_he3)
plt.title("window 32",color='black')

figure.add_subplot(2,2,3)
plt.imshow(local_eq3_he3)
plt.title("window 128",color='black')

figure.add_subplot(2,2,4)
plt.imshow(he3)
plt.title("he3",color='black')

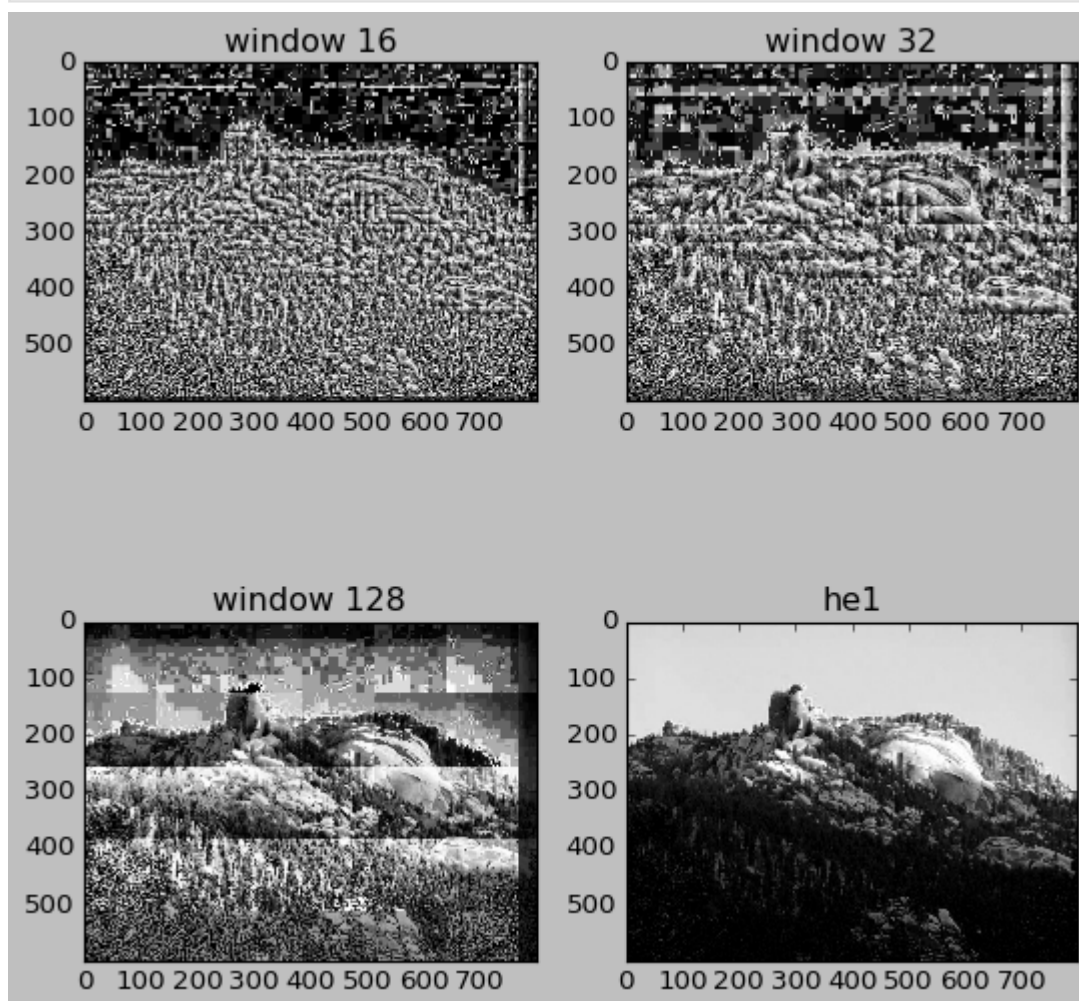
figure = plt.figure(figsize=(8,8))
figure.add_subplot(2,2,1)
plt.imshow(local_eq1_he4)
plt.title("window 16",color='black')

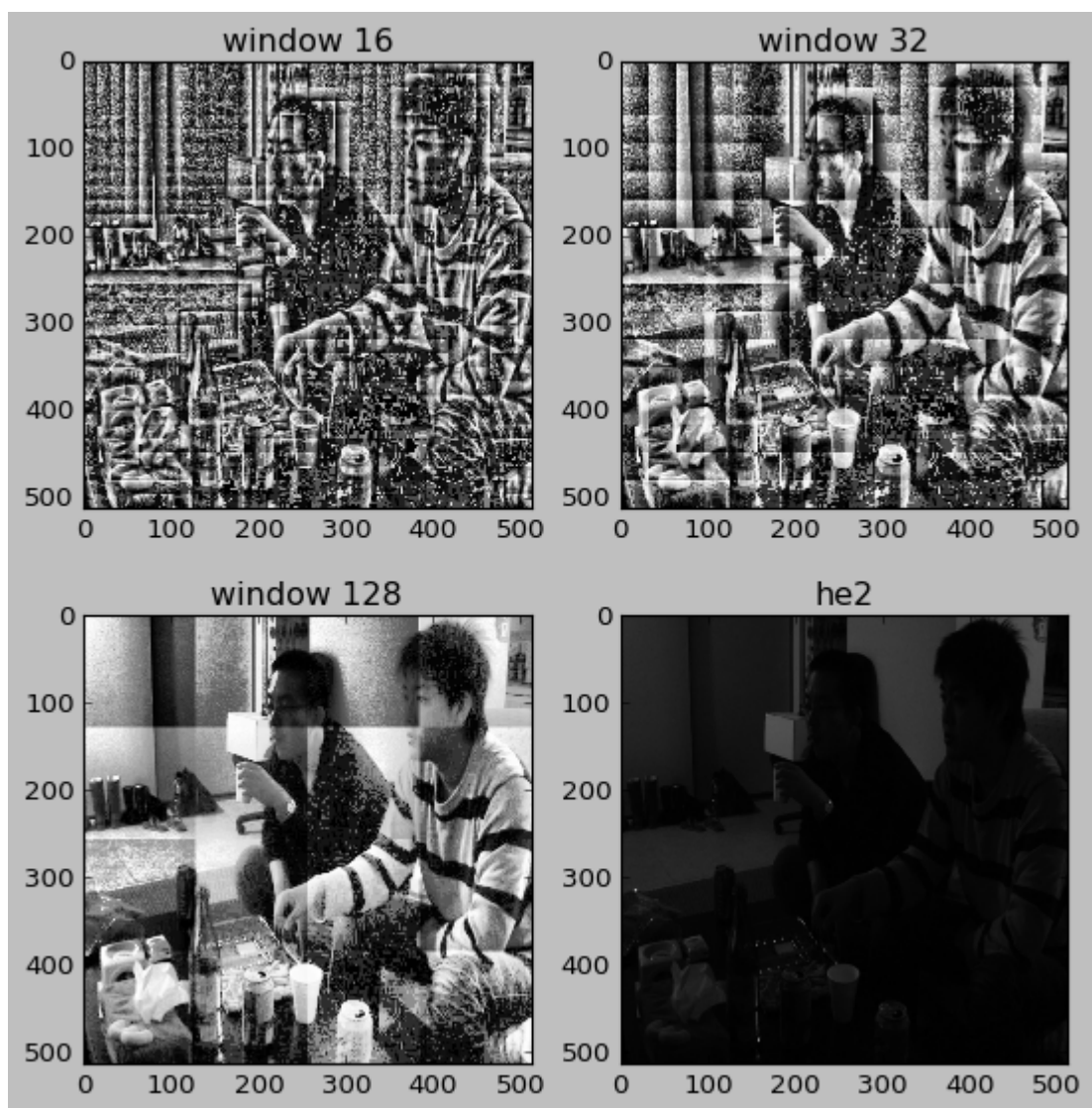
figure.add_subplot(2,2,2)
plt.imshow(local_eq2_he4)
plt.title("window 32",color='black')

figure.add_subplot(2,2,3)
plt.imshow(local_eq3_he4)
plt.title("window 128",color='black')

```

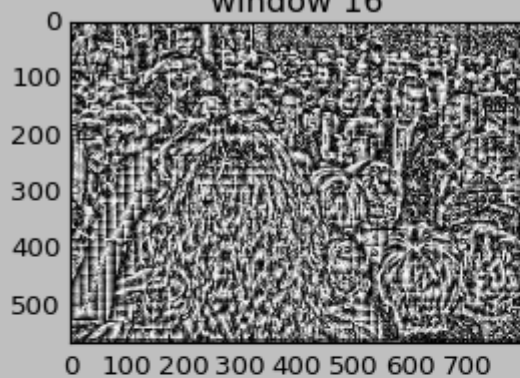
```
figure.add_subplot(2,2,4)
plt.imshow(he4)
plt.title("he4",color='black')
```



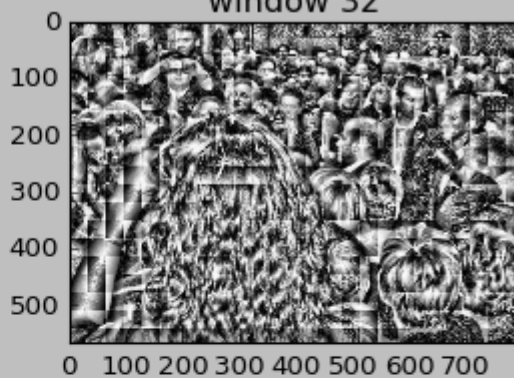


Out[]: Text(0.5, 1.0, 'he4')

window 16



window 32



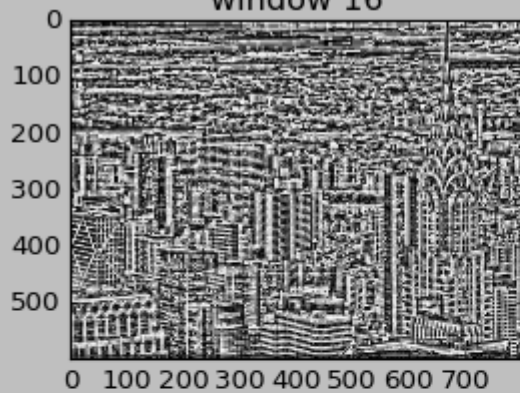
window 128



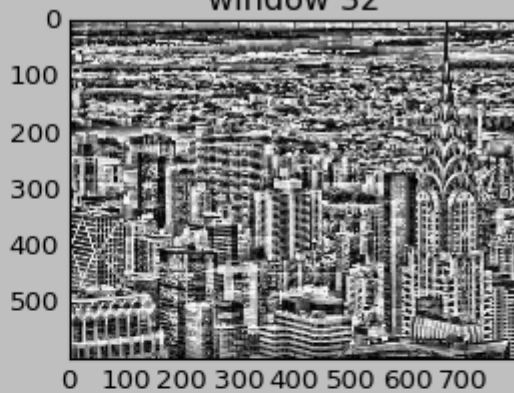
he3



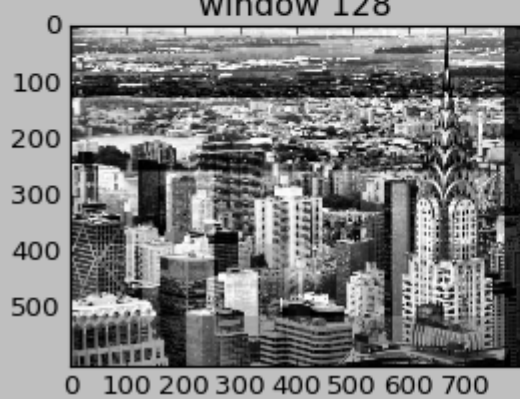
window 16



window 32



window 128



he4

